2.10 KNOWLEDGE BASE

In a mission-level combat model, decisions are based on the decision-maker's collection of knowledge which may be acquired through training, doctrine, and experience as well as other avenues. This part of the knowledge base contains tactical knowledge. Tactics tells the decision-maker what decision to make or action to perform (once values, thresholds, or conditions within the tactical situation are satisfied.) Decision-making systems acquire situational knowledge as a scenario unfolds. Situational knowledge tells the decision-maker when to think or act. At the start of the scenario, the weapons officer knows that he will fire a weapon when the target is at a specific relative bearing and range. However, the weapons officer does not know when, or even if, the criteria will be satisfied.

Situational knowledge is derived from direct and indirect perceptions. Direct perceptions are perceptions derived from sensors belonging to the player while indirect perceptions are perceptions obtained through communications with other players. Both types of perceptions and all other types of knowledge possessed by a player may be kept in long-term or short-term memory. Tactics normally belong in long-term memory; a player is not willing to discard them because they are not volatile and they are not easily replaced. Perceptions more likely belong in short-term memory because they become out-of-date if not updated.

2.10.1 Functional Element Design Requirements

The design requirements for the Knowledge Base functional element are:

a. Provide the user with the capability to build a player's tactical knowledge base. This data item will include specifications for evaluation rates, reactive movement, and resource allocation decisions.

b. Provide a capability for providing both friendly and enemy perceptions prior to model execution.

2.10.2 Functional Element Design Approach

Design Element 10-1: Tactical Knowledge Base

The tactical knowledge base for each player that will engage in thinking activities during a scenario is defined with the TDB TACTIC data item. The Suppressor core language instructions define evaluation rates; movement plans, resource allocation, and message reporting within the TACTIC data item. The user-provided information from the TDB is parsed to create a scenario player's tactical knowledge. During runtime, the tactical and situational knowledge is interwoven, and each scenario player's knowledge base is created.

A Suppressor player must have resource allocation tactics in order to react. The user creates resource allocation tactics using the RESOURCE-ALLOCATION data item. The types of actions defined by the RESOURCE-ALLOCATION data item are lethal assignment, nonlethal engagement, lethal engagement, movement, emission control, and change mode of control. For each of these action categories there are one or more procedures defined to control phases of the action. For example, for lethal engagement, there are six separate procedures. Two procedures define the conditions for which targets will be selected for possible engagement or dropped from consideration (LETHAL-ENGAGE-QUEUE-ADD and LETHAL-ENGAGE-QUEUE-DROP), two procedures define the conditions for starting and stopping the engagement (LETHAL-ENGAGE-START and LETHAL-ENGAGE-STOP), and the last two define the conditions for firing or stop firing an available weapon (LETHAL-ENGAGE-FIRING-START and LETHAL-ENGAGE-FIRING-STOP).

Each of the resource allocation procedures contain filtering and selection criteria that a player will use in choosing the resources to allocate to or deallocate from a perceived target. Exactly what resource is chosen is dependent on the particular procedure. The Suppressor core language currently allows over 70 tactical criteria for filtering in the resource allocation procedures. These criteria and additional details regarding the resource allocation procedures are presented in Section 2.11 for the Logic Processes FE.

Design Element 10-2: Create Player Perception

In Suppressor, all dynamic player decision-making is initiated by a player perception. There are several ways for a player to create a target perception. The simplest is to "pre-brief" the target to the player in the SDB by using a TOLD ABOUT data item. This item identifies the target and its location coordinates. The perception is created at time 0.0 of the scenario and behaves like any other kind of perception with one exception: it will never be dropped unless it can be determined that the target is dead. This is to allow the player to retain the target as a potential subject of engagement even if it hasn't been sensed for a particular amount of time.

The second way to create a perception is to receive a message about a target from another player. This can be done via intelligence or a target assignment from a commander. Receipt of such a message will cause a target perception to be created if it doesn't already exist.

Finally, the most common way to create a perception is to directly sense the target with a sensor system. The default situation is for a Suppressor player to create a perception of a target upon the first successful sensing chance. This can be overridden by use of the HITS-TO-ESTABLISH-TRACK data item in the sensor's CAPABILITY. The numeric value given for this input determines how many successful sensor chances (detections) must occur before the data is sent to the thinker system for perception creation. In addition, the input SCANS-IN-ESTABLISHING-TRACK can be specified to require an M "hits" out of N "scans" condition to be met.

For example, suppose that the HITS-TO-ESTABLISH-TRACK had a value of two and the SCANS-IN-ESTABLISHING-TRACK had a value of five. Then, a target perception could not be created until at least two successful sensor chances occurred within the most recent five chances.

It should be noted that the user can include a MAX-SNR-PERCEPTIONS data item in a TACTIC section of the perceiving player to limit the number of directly-sensed targets which can be retained as perceptions. The default without this data item is infinity.

2.10.3 Functional Element Software Design

Create Player Perception Module Design

The routine which handles the creation of perceptions is OBSTEL:

```
*begin logic to prepare sensor results report:
   *look up pointer to sensor/target chance;
   *when detection history (n-out-of-m) used to establish track:
      *incorporate this sense result into detection history;
      *determine detection criteria based on jamming;
      *determine number of hits in last m scans;
   *but, when detection history not used or no longer needed:
      *when target is visible:
         *look up rwr detection criteria;
         *when detection scheme not used (or not an rwr):
            *increment number of successful hits;
         *otherwise, rwr detection scheme used:
            *increment mainbeam hits if detection;
            *increment backlobe hits if detection;
         *end of test for rwr detection scheme.
      *end of test for target visible.
   *end of test for detection history.
   *when number of hits not met:
      *reset visible flag;
   *but, when number of hits for establishing track met:
      *set hits for both dry and jam cases;
      *when target is visible:
         *determine sensing rate (default to acquisition mode);
         *adjust rate for rwr mainbeam/backlobe criteria;
         *adjust rate depending on tracking and/or firing;
         *compute sensing cycle time;
         *when maximum coast time not defined:
            *when still linked to a thinker:
               *use perception drop time;
            *end of test if still linked to a thinker.
         *end of test for maximum coast time.
         *set detection history flag for coasting chances;
      *but, when sensor coasting:
         *increment number of sense chances when coasting;
         *when coast time exceeded:
            *reset successful detections counters;
            *reset detection history if needed;
         *end of test for coast time exceeded.
      *end of test for target visible.
   *end of test for number of hits met.
   *when any results to be reported:
      *allocate sensor results and store type of sensor;
      *allocate specific results and store results and limits;
      *allocate perceived cluster and initialize evaluation time;
      *store pointer to sensed acquisition/tracking results;
      *store perceived cluster ptr for all acq/track results;
      *link perception to sensor results;
      *allocate direct source and add to list;
      *allocate perceived location;
      *link perceived group list to perceived cluster;
      *store cluster id, interaction key ptr and sensing time;
      *when external sensor:
         *load perception with conf./decay factors;
      *end of test for external sensor.
      *look up flags and determine IFF result;
      *get raw sensor measurement data;
      *when a STD-DEV-MEASUREMENT-ERROR has been specified:
         *calculate true azimuth to target;
```

```
            *calculate elevation to target;
            *calculate 3-D range to target;
            *when azimuth measurement error specified:
               *introduce random error into azimuth;
            *end of test for azimuth measurement error specified.
            *when elevation measurement error specified:
               *introduce random error into elevation;
            *end of test for elevation measurement error specified.
            *when range measurement error specified:
               *introduce random error into range;
            *end of test for range measurement error specified.
            *recalculate X, Y, Z based on measurement errors:
         *but, when there is SEEKER-ERROR-DATA specified:
            *invoke logic to get terrain altitude;
            *invoke logic to get seeker off boresight errors;
         *end of test for STD-DEV- or SEEKER-ERROR data specified.
         *when planar (x,y) data can be obtained:
            *store (x,y) location data and set flag;
         *but, when azimuth to target can be obtained:
            *invoke logic to set terrain altitude if no error;
            *when sensor is above target and terrain:
               *calculate changes in x,y due to terrain differences;
               *store x,y location data, set planar location flag;
            *end of test for sensor above target and terrain.
         *end of test for planar data available.
         *store any observed altitude data, set flag;
         *store any observed speed, set flag;
            *calculate speed;
            *when speed measurement error specified:
               *introduce random error into speed;
            *end of test for speed measurement error specified.
         *when heading data can be obtained:
            *store heading data, set flag;
         *end of test for heading data.
         *when azimuth to target can be obtained:
            *store heading to target, set flag;
         *end of test for azimuth data.
         *store any observed player type in perception block;
         *when group types can be determined:
            *loop, until all group types are added:
               *find group on list and store type code;
            *end of loop for groups.
         *end of test for group type determination.
      *otherwise nothing to report:
         *recycle perceived groups;
         *recycle sensed acq/trk results;
      *end of test for anything to report.
   *end of logic for OBSTEL.
```

2.10.4 Assumptions and Limitations

- A players self-perception is always perfect.
- The experience of a player is not explicitly characterized in Suppressor. Think times and tactical evaluation rates are the same for all players of the same type.

2.10.5 Known Problems or Anomalies

None.